Výsledková listina sedmnáctého ročníku KSP po čtvrté sérii

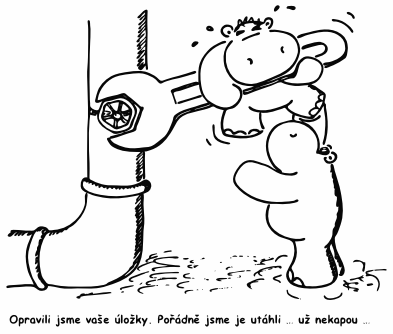
		škola	ročník	1741	1742	1743	1744	1745	suma	celkem
1.	Miroslav Klimoš	G Lanškr	0	10	10	8	10	14	52	186
2.	Josef Pihera	G Strakon	2	8	9	7	10	16	50	165
3.	Ondřej Bílka	G Zlín	3	1	9	8	10	13	41	153
4.	Peter Perešíni	GJGTajov	3		10	8	10		28	149
5.	Jan Pelc	G UBrod	3		8	4	10		22	135
6.	Zbyněk Konečný	GKptJaroš	2	10	10	4	4	10	38	131
7. – 8.	Peter Černo	GLŠtúra	4						0	130
	Pavel Klavík	G Chrudim	2	1	10	7	10	13	41	130
9.	Miroslav Cicko	GJGTajov	4						0	123
10.	Adam Zivner	G UBrod	3	2	8	5	10	12	37	120
11.	Jan Bulánek	G Klatovy	4		10	8	10		28	99
12.	Martin Koníček	G UBrod	4		6	4	2	10	22	86
13.	Jakub Kaplan	GJKTyla	1		8	5	4	13	30	83
14.	Roman Smrž	GOhradní	1	8	10	1	10	12	41	79
15.	Jan Hrnčíř	GFXŠaldy	3	8	1	4	4		17	77
16.	Lukáš Lánský	GJKTyla	1		7	0	4	10	21	66
17.	Petr Kratochvíl	G SvětláNS	2	1	5	4	3		13	61
18.	Eva Schlosáriková	G Piešťany	4	8	7	4	4		23	59
19.	Martin Čech	G UBrod	4						0	52
20.	Stanislav Basovník	G Kroměříž	4						0	43
21.	Tomáš Herceg	G Třebíč	2	1	5	4			10	42
22.	Cyril Hrubiš	G Bílovec	3						0	40
23.	Martin Kupec	GMendel	3	1	4				5	33
24.	Zbyněk Falt	GNeumannov	4						0	32
25.	Daniel Marek	GZborov	3	1	9	7	10		27	27
26.	Josef Špak	GJírovco	2						0	25
27. – 28.	Tereza Klimošová	G Lanškr	3		10			14	24	24
	Lukáš Špalek	G Čadca	4						0	24
29.	Michal Pavelčík	G UBrod	2						0	20
30. – 31.	Ondřej Bouda	GKptJaroš	2						0	18
	Adam Ráž	GBudějo	2						0	18
32.	Marian Kaluža	GHavlíčkov	2						0	16
33. – 35.	Jiří Cabal	SPŠ DvKrál	2						0	15
	Ondřej Garncarz	G Příbor	4						0	15
	Martin Kahoun	GJNerudy	2						0	15
36.	Jan Palenčar	G Martin	2						0	14
37.	Martin Podloucký	G Strážnic	4						0	12
38. – 41.	Jakub Jenis	GsvCyrMet	1						0	11
	Hana Klempová	GUBalvanJN	4						0	11
	Jakub Porod	G Týn nV	2						0	11
	Ján Zahornadský	GZborov	4						0	11
42. – 44.	Lukáš Beleš	G Čadca	4						0	10
	Jakub Benda	GJNerudy	2						0	10
	Michal Vaner	G Turnov	3						0	10
45. – 47.	Kateřina Böhmová	G Rožnov	3	4			4		8	8
	Jiří Machálek	G Holešov	3						0	8
	Petr Soběslavský	GJHeyrovs	4						0	8
48. – 49.	Daniel Sedláček	SPŠE Hav	1						0	7
	Filip Šauer	G Klatovy	4						0	7
50.	Jiří Nohavec	G Domažl	4						0	6
51. – 54.	Dalibor Adamčík	SPŠE Preš	2						0	5
	Petr Musil	G MBuděj	3						0	5
	Jan Staněk	GKptJaroš	3						0	5
	Zdeněk Vilušínský	G Turnov	4						0	5
55. – 56.	Tomáš Ehrlich	G Holešov	2						0	2
	Martin Vařák	G Bílovec	2						0	2
57. – 59.	Florián Danko	SPŠEtech	2						0	1
	Tamara Kuštárová	GBiling	0						0	1
	Petr Zimčík	G UBrod	1						0	1
60. – 62.	Miroslav Hovorka	GJateční	4						0	0
	Adrián Lachata	G Svidník	3						0	0
	Michal Onderko	SPŠ Karviná	3			0			0	0

Milí řešitelé!

Do ruky se Vám dostává opravená už čtvrtá série našeho semináře. Nicméně nevzdychejte nad jejími výsledky a pusťte se chutě do řešení páté, letos poslední série, jejíž zadání Vám už přišlo s opravenou sérií třetí. Pro jistotu připomínáme, že termín odeslání páté série je 2. května 2005 a řešení můžete odevzdávat jak elektronicky na *http://ksp.mff.cuni.cz/submit/*, tak klasickou poštou na známou adresu:

Korespondenční seminář z programování
KSVI MFF UK
Malostranské náměstí 25
Praha 1, 118 00

Aktuální informace o KSP naleznete na stránkách *http://ksp.mff.cuni.cz/*, dotazy organizátorům můžete posílat e-mailem na adresu *ksp@mff.cuni.cz*.



Vzorová řešení čtvrté série sedmnáctého ročníku KSP

17-4-1 Mandarinková zeď

V některých paralelních vesmírech císař *No-san* zkrachoval, či nepřežil povstání svých nevěrných poddaných, ale jinde jeho Mandarínie dále prosperovala díky Vašim radám.

Problém můžeme rozdělit na dva případy. Pokud je strážců sudý počet, je řešení jednoduché. Označme si $P(i)$ počet medailí, které vyžaduje i-tý strážce. Celkem nám bude stačit maximum z požadavků libovolných dvou sousedů – $p_m = \max(P(i) + P(i+1))$, přičemž indexy bereme cyklicky, takže $n + 1 = 1$. Druhy medailí budeme označovat čísly $1..p$. Medaile rozdělíme takto: Strážci na liché pozici dáme medaile $1..P(i)$, strážci na sudé pozici dáme medaile $p_m - P(i) + 1..p_m$. Každí dva sousedi se liší paritou pozice a proto mají dohromady medaile $1..P(i), p_m - P(i)+1..p_m$ a určitě nemají žádnou oba dva, protože pak by jich měli dohromady více než p_m.

Podívejme se na lichá n. Určitě potřebujeme alespoň p_m medailí, ale můžeme jich potřebovat i více. Například třem strážcům musíme dát tolik medailí, kolik je součet jejich požadavků. Označme si $S(i) = \sum_{k=1}^{i} P(i)$ součet prvních i požadavků. Protože jeden druh medaile můžeme dát maximálně pouze m strážcům, kde $n = 2m + 1$, budeme určitě potřebovat alespoň $p_s = \lceil S(n)/m \rceil$ medailí. Ukážeme, že nám bude vždy stačit $p = \max(p_s, p_m)$ medailí.

Myšlenka je asi taková, že máme na začátku množinu medailí L, které má strážce na liché pozici, a medaile P, které má strážce na sudé pozici. Protože ale poslední strážce je na liché pozici, měly by se množiny v průběhu rozdělování prohodit tak, aby poslední strážce měl jiné medaile než ten první. Medaile budeme rozdělovat speciálním způsobem. Půjdeme od prvního strážce k poslednímu a přitom jim budeme dávat medaile. Zapíšeme si medaile do nekonečné cyklické posloupnosti $1..p, 1..p, \ldots$ a budeme je přiřazovat strážcům popořadě. Označíme tuto posloupnost a, $a[i] = ((i-1) \bmod p) + 1$. Můžeme tedy explicitně zapsat, jaké medaile dostane i-tý strážce – $a[S(i-1)+1]..a[S(i)]$. Protože $p \geq p_m$, nemohou dostat žádní dva sousedé stejné medaile.

Takto rozdělujeme medaile, ale jen do té doby, dokud mají strážci na liché pozici alespoň 1 z medailí $1..P(1)$ prvního strážce. Hledáme tedy *nejmenší* k takové, že součet požadavků do k-tého lichého strážce $S(2k+1)$ je menší nebo roven $k \cdot p$. Že takové k existuje se můžeme například přesvědčit tak, že zvolíme $k = m$. Pak víme, že $mp \geq m \cdot p_s = m \cdot \lceil S(n)/m \rceil \geq S(n)$, takže víme, že pro $k = m$ je předpoklad splněn a vždy takové k existuje. Nyní si ukážeme, že pokud rozdělíme výše popsaným způsobem medaile prvním $2k$ strážcům, můžeme strážcům $2k+1..n$ dávat medaile už podle parity jako pro n sudé. Podívejme se na to, jaké medaile dostane strážce na pozici $2k$. Protože k je minimální, $S(2k-1) > (k-1) \cdot p$, proto strážce na pozici $2k$ nemá žádnou z barev $p..p - P(2k+1) + 1$ (nakreslete si obrázek). Jsme tedy schopni najít rozdělení pro p druhů medailí, ale nám stačí jenom tento počet.

Algoritmus bude velmi jednoduchý, spočteme p_m a pro n sudé vypíšeme tuto hodnotu, pro n liché si ještě spočítáme p_s a vrátíme tu větší z nich. To vše určitě zvládneme s lineárním časem i pamětí – tedy $O(n)$.

Petr Škoda

17-4-2 Válicie

Úlohu rozmístit Válicie na křižovatkách Mandarínie tak, aby na každé křižovatce byla právě jedna, převedeme na úlohu barvení vrcholů grafu dvěma barvami. Lze snadno nahlédnut, že obarvení prvního vrcholu jednou ze dvou barev (postavit nebo nepostavit stanici) určuje barvu ostatních vrcholů, které jsou s ním spojeny cestou. Vrcholy vzdálené o lichý počet jednotek nesmí být obarvené stejnou barvou, na rozdíl od těch na sudých pozicích, které ji musí mít stejnou. Z této úvahy hned plyne, že dvěma barvami nelze obarvit graf, který obsahuje cyklus liché délky (například trojúhelník). Protože potřebujeme minimalizovat počet stanic, vybereme si v každé komponentě souvislosti grafu tu barvu, kterou je obarveno méně vrcholů.

Algoritmus řešící úlohu může být následující. Vezmeme vrchol, obarvíme ho, a všechny jeho dosud neobarvené sousedy obarvíme týmž algoritmem druhou barvou. Úloha nemá řešení, pokud nějaký soused zpracovávaného vrcholu má již stejnou barvu. V tom případě totiž existuje v grafu cyklus liché délky.

Po dokončení obarvování komponenty zkontrolujeme, jestli je barvy značící „postavit stanici“ méně než barvy druhé. Pokud ne, barvy v komponentě zinvertujeme.

Vrcholy jsou obarvované rekurzivní funkcí pracující se seznamem sousedů. Každý vrchol je zpracován nanejvýš dvakrát. Proto je časová i paměťová složitost $O(N + M)$.

Miroslav „miEro“ Rudišin

17-4-3 Phirma

Nejveleváženější císaři No-sane!

Dle Tvého hlubokomyslného rozkazu Ti phirma Jakobi-Čestná zasílá skvostné dary. Na stanovení její ceny se podíleli nejpovolanější učenci, slovutní programátoři a osvícení teoretici, kteří za použití nejďábelštějších, nejfantasknějších a nejdůmyslnějších konstrukcí, mohutného binárního stromoví roztodivných názvů, jakož i větvoví intervalového a AVL, namnoze pak půlení binárního, sestavili vzletné programy lepých tvarů.

S nejuctivějšími pozdravy
Skutečně-Nečestná, účetní

Vážená paní Skutečně-Nečestná,

již jsme se chystali Váš velkolepě vyhlížející dar přijmout, když tu jakýsi účetní nevelkých znalostí povšiml si výpočtu mnohem jednoduššího, nemnoha struktur vyžadujícího, prabídně prostého, ba až hanebně rychlého.

Poslyšte návod:

Většinu řešitelů napadla jednoduchá myšlenka – vytvořit k zadané posloupnosti $a_1, \ldots, a_N$ posloupnost částečných součtů $s_1, \ldots, s_N$, kde $s_i = \sum_{k=1}^{i} a_k$, a řešení pak hledat prostým prozkoumáním všech možných dvojic. Takový postup je sice průzračný a zaručeně vede k výsledku, ale trvá $O(N^2)$. Někteří ostřílení řešitelé objevili, že různými hrátkami se stromy můžeme vyzískat řešení v čase $O(N \log N)$, ale my si ukážeme řešení v čase $O(N)$.

Chceme tedy najít dvojici indexů i a j ($i \leq j$) takovou, aby $s_j - s_{i-1} = a_i + \ldots + a_j > 0$ a $j - i$ bylo nejvyšší možné. Navíc máme vybrat úsek s největším součtem. Využijeme nápadu s posloupností částečných součtů s_i. Dále si připravíme pomocnou strukturku – posloupnost $m_1, \ldots, m_N$, kde $m_i = max(s_i, \ldots, s_N)$ a indexy i a j, které nastavíme na začátek posloupnosti.

V každém kroku se snažíme najít nejdelší kladný úsek, který začíná prvkem i, ale děláme to jenom tehdy, když máme jistotu, že může být výhodnější než zatím nejdelší nalezený úsek. Index j tedy posouváme tak dlouho, dokud platí, že $s_i < m_{j+1}$. Dále už nesmíme j zvyšovat, protože m_j je maximem z prvků $s_j, \ldots, s_N$, takže za indexem j se částečné součty už jenom snižují (to bychom si k zatím nalezenému kladnému úseku přičítali záporné prvky).

Jakmile nalezneme poslední j, pro které ještě platí $m_j > s_i$, posuneme index i na $i+1$ a zkusíme najít nový kladný úsek. Klíčovým pozorováním je fakt, že s indexem j se nemusíme vracet, zlepšení může přinést jedině posun dále. Kdybychom se s j vrátili zpět, můžeme už získat jenom kratší úsek než ten už dříve nalezený.

Kdykoliv nalezneme nový úsek s kladným součtem prvků, porovnáme ho se zatím nejlepším nalezeným úsekem a zapamatujeme si samozřejmě ten lepší z nich. Porovnáváme nejprve podle délky, v případě shody ještě podle součtu prvků (ten můžeme počítat v konstantním čase, protože $a_i + \ldots + a_j = \sum_{k=1}^{j} a_k - \sum_{k=1}^{i-1} a_k = s_j - s_{i-1}$).

Jak i, tak j projdou posloupnost nejhůř jednou od začátku do konce, a protože načíst a vytvořit všechna pole umíme v čase $O(N)$, má algoritmus lineární časovou složitost.

Jana Kravalová

17-4-4 Antifňákovník

Celkem jednoduché řešení této úlohy bylo v čase $O(n^2)$ vyzkoušet všechny dvojice kabelů vlevo a vpravo. Pro trochu složitější řešení si všimnu, že můžu položit dotaz „které pravé kabely jsou připojeny k těmto levým“ v čase $O(n)$. V takovém čase si stihnu nalevo připojit k zemnění ty kabely, které potřebuji, a zjistit, které z pravých jsou uzemněny.

Nyní si stačí vybrat vhodné podmnožiny kabelů nalevo. Kabely si očísluji $1..n$ a budu zkoumat, jaké kabely pasují ke kabelům vpravo – R_i je číslo toho kabelu nalevo, který je spojen s kabelem i vpravo. Vyberu-li nalevo nejprve kabely s číslem nedělitelným dvěma, budou mít všechny odpovídající kabely vpravo určitě R_i liché, zatímco ostatní jsou buď nezapojené nebo mají R_i sudé. Takto jsem vlastně zjistil, jak bude vypadat 0. bit čísel $R[i]$. A stejně mohu zjistit i 1., 2., ..., ($\log n$)-tý bit: zapojím vždy kabely s i-tým bitem nenulovým a nastavím tento bit odpovídajícím kabelům v R_i, čili kabelům, jejichž levý konec je připojen na zemnění a má i-tý bit nenulový. Pokud bude mít nakonec nějaký kabel $R_i = 0$, pak je určitě nezapojený, jinak bude v R_i číslo odpovídajícího levému kabelu.

Toto řešení má časovou složitost $O(n \log n)$. Navíc je to nejmenší možná složitost, což můžeme dokázat takto: i kdyby byly všechny kabely zaručeně propojeny, potřebuji zjistit, kterou z permutací mám před sebou. Těch je $n!$, potřebuji tedy získat řádově $\log(n!) \approx n \log n$ bitů, přičemž jednou odpovědí získám právě 1 bit. Toto je tedy dolní odhad slabší verze našeho problému, s nezapojenými kabely je to určitě jen složitější.

Tomáš Gavenčiak

17-4-5 Jazykozpytec vrací úder

Odvážnému štěstí přeje, praví se, a velmi podobně tomu bylo i ve čtvrté seriálové úloze. Kdo v sobě našel dosti odvahy přečíst si dlouhé a hrozivě vypadající zadání, a pochopit, co se po něm vlastně chce, zjistil, že všechny úlohy jsou velmi snadné. O tom koneckonců svědčí i bodové zisky. Účelem tentokrát nebylo vymýšlet komplikované algoritmy na komplikované problémy, jako si spíše přesně uvědomit, jak spolu souvisí různé druhy dosud převedených automatů. Ale teď už k správným řešením.

Úloha 1: Chceme-li ukázat, že jazyk $L = \{0^n 1^m;\ 1 \leq n \leq m\}$ lze rozpoznávat deterministickým zásobníkovým automatem koncovým stavem, zkrátka takový automat sestrojíme. DZA bude používat stavy $Q = \{l, p, f\}$, zásobníkové symboly $Z = \{z, 0\}$, počáteční stav bude l a počáteční zásobníkový symbol z, jediný přijímací stav bude f. Sada instrukcí bude následující:

$\delta(l, 0, z) = (l, z0)$ ... načti první 0
$\delta(l, 0, 0) = (l, 00)$ ... čti další 0
$\delta(l, 1, 0) = (p, \lambda)$ ... začni odmazávat 0 ze zásobníku
$\delta(p, 1, 0) = (p, \lambda)$ ... maž další 0
$\delta(p, \lambda, z) = (f, z)$ ... už máme $0^n 1^n$
$\delta(f, 1, z) = (f, z)$ ... dočítej zbylé 1

Princip je stejný jako u příkladu v zadání s tím rozdílem, že při načtení $0^n 1^n$ se ještě načítá libovolný počet 1. Kdyby se při dočítání vyskytla 0, stroj se zastaví na nedefinovanou instrukci, a jelikož se nenačetlo slovo celé, bude odmítnuto.

Chceme-li ukázat, že DZA přijímajícím prázdným zásobníkem jazyk L nemůže nikdy rozpoznávat, budeme argumen-

Úloha 17-4-3 – Phirma – program

```
#include <stdio.h>
#define MAX (a, b) ( ( (a) > (b)) ? (a) : (b))
#define MAX_N 10000

int a[MAX_N+1];                                   /* zadaná posloupnost */
int s[MAX_N+1];                                   /* posloupnost částečných součtů */
int m[MAX_N+1];                                   /* pomocná posloupnost */
int N;

int main (void) {
  int i, j;
  int left, right;                                /* hranice zatím nejlepšího úseku */

  printf ("N:");
  scanf ("%d", &N);

  a[0]=s[0]=0;
  for (i=1; i<=N; i++) {
    printf ("%d. člen:", i); scanf ("%d", a+i);   /* načtení zadané posloupnosti */
    s[i]=s[i-1]+a[i];                             /* vytvoření posloupnosti částečných součtů */
  }

  m[N]=s[N];
  for (i=N-1; i>=1; i--) m[i]=MAX (m[i+1], s[i]); /* vytvoření pomocné posloupnosti */

  left=0; right=0;
  for (i=j=0; j<N; i++) {
    while (j<N && m[j+1]>s[i]) j++;               /* najdeme novou pravou hranici */
    if (j-i < right-left || s[j]-s[i] < s[right]-s[left]) continue;   /* kratší nebo s menším součtem */
    left=i;
    right=j;
  }

  if (left-right) {
    printf ("Hledaný úsek délky %d je␣", right-left);
    for (i=left+1; i<=right; i++) printf ("%d␣", a[i]);
    putchar ('\n');
  } else printf ("Žádný hledaný úsek neexistuje.\n");

  return 0;
}
```

Úloha 17-4-4 – Antifňákovník – program

```
function testuj(i:integer):boolean;
{jen se zeptá zda je kabel i vpravo právě zapojen}
var c:char;
begin
  write('?',i,' ');readln(c);testuj:=c='a';
end;

var
  i,j,N:integer;
  r:array[1..N] of integer;
begin
  readln(N);
  for i:=0 to trunc(log2(N)) do
  begin
    for j:=1 to N do
      if (j and (1 shl i))<>0 then writeln('+',j);
    for j:=1 to N do
      if testuj(j) then r[j]:=r[j] or (1 shl i);
    for j:=1 to N do
      if (j and (1 shl i))<>0 then writeln('-',j);
  end;
  for j:=1 to N do
    if r[j]<>0 then writeln(r[j],'->',j);
end.
```

```
    else
    begin
      S:= 0;
      for I:= 1 to N do
        S:= S + P[I];

      Ps:= (2*S + N - 2) div (N - 1);
      if Ps > Pm then Writeln(Ps) else Writeln(Pm);
    end;
  end;
end.
```

Úloha 17-4-2 – Válicie – program

```
#include <stdio.h>
#define MAX_N 1000
#define ABS (a)  ( ( (a) < 0)  ? - (a) :  (a))

int sousedi[MAX_N+1][MAX_N+1];                  /* mělo by se dynamicky alokovat; každý to zvládne */
int sousedi_len[MAX_N+1];
int barvy[MAX_N+1];                             /* barva vrcholů */
int nvrcholu[2],  nstanic;                      /* počet vrcholů dané barvy a stanic celkem */

int obarvi (int v,  int barva) {
  barvy[v] = barva;                             /* >0 – se stanicí, <0 – bez stanice */
  nvrcholu[  (barva > 0)  ? 0 : 1 ]++;

  for  (int i=0; i<sousedi_len[v]; i++) {
    if  ( barvy[ sousedi[v][i] ] == barva ) return 0;   /* lichý cyklus */

    if  ( ABS (barvy[ sousedi[v][i] ]) < ABS (barva) )  /* pokud ještě nebyl obarven v tomto kole, přebarvi */
      if  (!obarvi (sousedi[v][i],  -barva)) return 0;
  }
  return 1;
}

int main () {
  int i,  n,  m,  v[2];

  printf ("Zadejne n a m:");
  scanf ("%d %d", &n, &m);                      /* počet měst a cest */
  for  (i=0; i<m; i++) {
    printf ("Zadejte %d. hranu:", i+1);
    scanf ("%d %d", v+0, v+1);
    sousedi[v[0]][sousedi_len[v[0]]++]=v[1];
    sousedi[v[1]][sousedi_len[v[1]]++]=v[0];
  }
  for  (i=1; i<=n; i++)
    if  (barvy[i] == 0) {                       /* chceme pouze neobarvené */
      nvrcholu[0] = nvrcholu[1] = 0;            /* počet barev v komponentě */
      if  (!obarvi (i,  1))                     /* zkusime začít 'kladnou' barvou */
        break;
      nstanic += nvrcholu[  (nvrcholu[0] < nvrcholu[1])  ? 0 : 1 ];
      if  (nvrcholu[0] > nvrcholu[1]) obarvi (i,  -2);   /* lze umístnit méně stanic, přebarvíme */
    }
  if  (i > n) {                                 /* povedlo se obarvit všechny vrcholy */
    printf ("Stačí postavit %d stanic.\n", nstanic);
    for  (i=1; i<=n; i++) if  (barvy[i] > 0) printf ("%d\n", i);
  } else printf ("Stanice postavit nelze!\n");

  return 0;
}
```


tovat takto: Kdybychom takový automat měli, slovo 0^n1^m by bylo přijato, tedy automat by vyprázdnil zásobník a zastavil se tak. Ale slovo 0^n1^{m+1} tím pádem už nikdy nemůže být rozpoznáno, protože automat se zastavil už o krok dříve. Proto žádný takový stroj nemůže vůbec existovat. Podobně lze najít i regulární jazyk nerozpoznatelný DZA-PZ. Bude to třeba jazyk $\{a^i;\ i \in N\}$ (proč je regulární snad již nemusíme zdůvodňovat) a použijeme téměř stejný argument.

Úloha 2: U jednoho směru převodu, tedy NZA přijímající prázdným zásobníkem na ekvivalentní NZA přijímající koncovým stavem, projde naprosto stejný postup jako u deterministických ZA, který jsme si předvedli v zadání. Druhý směr je zajímavější a již nutně musí nějak využívat nedeterminismu stroje. Mějme tedy nějaký NZAKS $M = (Q, A, Z, \delta, q_0, z_0, F)$. Do M přidáme speciální stav q_f a instrukce $\delta(q_f, \lambda, z) = (q_f, \lambda)$ pro každý zásobníkový symbol z. Kdykoli se vstoupí do stavu q_f, zásobník se vymaže a stroj se zastaví. Z každého přijímacího stavu f potom natáhneme „odbočku“ do stavu q_f pomocí instrukcí $\delta(f, \lambda, z) = (q_f, z)$ pro každý $z \in Z$ a $f \in F$. V každém přijímacím stavu se pak stroj nedeterministicky rozhodne, jestli už je to ten opravdu poslední stav (neboli slovo je celé načteno), v tom případě odbočí a přijme slovo prázdným zásobníkem. Pokud by výpočet odbočil dříve než je celé slovo přečteno, podle naší definice zásobníkového automatu bude slovo odmítnuto a nebudou tak přijímány nesmysly. Zjevně jsme tak tedy sestrojili ekvivalentní automat přijímající prázdným zásobníkem.

Úloha 3: Rozmysleme si ještě pro pořádek, jak se dá u automatů pohlížet na nedeterminismus. První úhel pohledu je takový, že stroj, může-li se v některých situacích nedeterministicky rozhodnout, je veden neomylnou intuicí, a vždy si vybere tu možnost, která vede k přijetí slova. Pokud žádná cesta k přijetí neexistuje, pak ani neomylná intuice nepomůže, a slovo bude odmítnuto. Druhý pohled je ten, že nedeterministický automat je schopen paralelně provádět mnoho větví výpočtu, a tak najít cestu k přijetí, pokud taková ovšem vůbec existuje. Oba pohledy vystihují naši původní definici nedeterminismu.

Myšlenka nedeterministického přijímání jazyka všech palindromů nad abecedou $A = \{a, b\}$ je poměrně jednoduchá: budeme načítat symboly na zásobník, nedeterministicky uhodneme, jestli právě přišel střed palindromu, načež začneme zásobník vyprazdňovat a kontrolovat, jestli jsou obě poloviny symetrické. Sepišme si to přesně.

Zkonstruujeme NZA přijímající prázdným zásobníkem, jehož množina stavů bude $Q = \{l, p\}$, zásobníkové symboly $Z = \{z_0, a, b\}$, počáteční stav bude l a počáteční zásobníkový symbol z_0. Sada instrukcí bude tato:

$\delta(l, x, z) = (l, zx)\ \forall x \in A, z \in Z$ ... načti levou půlku

$\delta(l, x, z) = (p, zx)\ \forall x \in A, z \in Z$ ... uhodni *sudý* střed

$\delta(l, x, z) = (p, z)\ \forall x \in A, z \in Z$ ... uhodni *lichý* střed

$\delta(p, x, x) = (p, \lambda)\ \forall x \in A$ ... ověř pravou půlku

$\delta(p, \lambda, z_0) = (p, \lambda)$ ... vyprázdni zásobník a skonči

Zjevně pokud automat uhodl střed na správné pozici a obě půlky byly symetrické, slovo bude přijato. Pokud symetrické nebyly, stroj se zastaví před vyprázdněním zásobníku na nedefinovanou instrukci. Pokud stroj uhodl střed palindromu příliš brzo či příliš pozdě, nebude načteno celé slovo nebo se zastaví se na nedefinovanou instrukci před vyprázdněním zásobníku. Tyto větve výpočtu tedy nevydají špatný výsledek a automat tak skutečně přijímá právě jazyk všech palindromů.

Že na tento jazyk nestačí DZA přijímající prázdným zásobníkem se dá odůvodnit téměř stejně, jako v první úloze. Kdyby takový automat přijal palindrom a^i, nemohl by již přijmout palindrom a^{i+1}.

A to je celé.

Tomáš Valla

Úloha 17-4-1 – Mandarinková zeď – program

```
program MandarinkovaZed;
const
  MaxN = 1000;
var
  P: array[1..MaxN] of Integer;
  N: Integer;

  I, Pm, Ps, S: Integer;
begin
  Readln(N);
  for I:= 1 to N do
  Read(P[I]);

  if N = 1 then
    Writeln(P[1])
  else
  begin
    Pm:= P[1] + P[N];
    for I:= 1 to N - 1 do
    if P[I] + P[I + 1] > Pm then
      Pm:= P[I] + P[I + 1];

    if N mod 2 = 0 then
      Writeln(Pm)
```

```
    else
    begin
      S:= 0;
      for I:= 1 to N do
        S:= S + P[I];

      Ps:= (2*S + N - 2) div (N - 1);
      if Ps > Pm then Writeln(Ps) else Writeln(Pm);
    end;
  end;
end.
```

Úloha 17-4-2 – Válicie – program

```
#include <stdio.h>
#define MAX_N 1000
#define ABS (a)  ( ( (a) < 0)  ? - (a) :  (a))

int sousedi[MAX_N+1][MAX_N+1];                  /* mělo by se dynamicky alokovat; každý to zvládne */
int sousedi_len[MAX_N+1];
int barvy[MAX_N+1];                             /* barva vrcholů */
int nvrcholu[2],  nstanic;                      /* počet vrcholů dané barvy a stanic celkem */

int obarvi (int v,  int barva) {
  barvy[v] = barva;                             /* >0 – se stanicí, <0 – bez stanice */
  nvrcholu[  (barva > 0)  ? 0 : 1 ]++;

  for  (int i=0; i<sousedi_len[v]; i++) {
    if  ( barvy[ sousedi[v][i] ] == barva ) return 0;   /* lichý cyklus */

    if  ( ABS (barvy[ sousedi[v][i] ]) < ABS (barva) )  /* pokud ještě nebyl obarven v tomto kole, přebarvi */
      if  (!obarvi (sousedi[v][i],  -barva)) return 0;
  }
  return 1;
}

int main () {
  int i,  n,  m,  v[2];

  printf ("Zadejne n a m:");
  scanf ("%d %d", &n, &m);                      /* počet měst a cest */
  for  (i=0; i<m; i++) {
    printf ("Zadejte %d. hranu:", i+1);
    scanf ("%d %d", v+0, v+1);
    sousedi[v[0]][sousedi_len[v[0]]++]=v[1];
    sousedi[v[1]][sousedi_len[v[1]]++]=v[0];
  }
  for  (i=1; i<=n; i++)
    if  (barvy[i] == 0) {                       /* chceme pouze neobarvené */
      nvrcholu[0] = nvrcholu[1] = 0;            /* počet barev v komponentě */
      if  (!obarvi (i,  1))                     /* zkusime začít 'kladnou' barvou */
        break;
      nstanic += nvrcholu[  (nvrcholu[0] < nvrcholu[1])  ? 0 : 1 ];
      if  (nvrcholu[0] > nvrcholu[1]) obarvi (i,  -2);   /* lze umístnit méně stanic, přebarvíme */
    }
  if  (i > n) {                                 /* povedlo se obarvit všechny vrcholy */
    printf ("Stačí postavit %d stanic.\n", nstanic);
    for  (i=1; i<=n; i++) if  (barvy[i] > 0) printf ("%d\n", i);
  } else printf ("Stanice postavit nelze!\n");

  return 0;
}
```


tovat takto: Kdybychom takový automat měli, slovo 0^n1^m by bylo přijato, tedy automat by vyprázdnil zásobník a zastavil se tak. Ale slovo 0^n1^{m+1} tím pádem už nikdy nemůže být rozpoznáno, protože automat se zastavil už o krok dříve. Proto žádný takový stroj nemůže vůbec existovat. Podobně lze najít i regulární jazyk nerozpoznatelný DZA-PZ. Bude to třeba jazyk $\{a^i;\ i \in N\}$ (proč je regulární snad již nemusíme zdůvodňovat) a použijeme téměř stejný argument.

Úloha 2: U jednoho směru převodu, tedy NZA přijímající prázdným zásobníkem na ekvivalentní NZA přijímající koncovým stavem, projde naprosto stejný postup jako u deterministických ZA, který jsme si předvedli v zadání. Druhý směr je zajímavější a již nutně musí nějak využívat nedeterminismu stroje. Mějme tedy nějaký NZAKS $M = (Q, A, Z, \delta, q_0, z_0, F)$. Do M přidáme speciální stav q_f a instrukce $\delta(q_f, \lambda, z) = (q_f, \lambda)$ pro každý zásobníkový symbol z. Kdykoli se vstoupí do stavu q_f, zásobník se vymaže a stroj se zastaví. Z každého přijímacího stavu f potom natáhneme „odbočku“ do stavu q_f pomocí instrukcí $\delta(f, \lambda, z) = (q_f, z)$ pro každý $z \in Z$ a $f \in F$. V každém přijímacím stavu se pak stroj nedeterministicky rozhodne, jestli už je to ten opravdu poslední stav (neboli slovo je celé načteno), v tom případě odbočí a přijme slovo prázdným zásobníkem. Pokud by výpočet odbočil dříve než je celé slovo přečteno, podle naší definice zásobníkového automatu bude slovo odmítnuto a nebudou tak přijímány nesmysly. Zjevně jsme tak tedy sestrojili ekvivalentní automat přijímající prázdným zásobníkem.

Úloha 3: Rozmysleme si ještě pro pořádek, jak se dá u automatů pohlížet na nedeterminismus. První úhel pohledu je takový, že stroj, může-li se v některých situacích nedeterministicky rozhodnout, je veden neomylnou intuicí, a vždy si vybere tu možnost, která vede k přijetí slova. Pokud žádná cesta k přijetí neexistuje, pak ani neomylná intuice nepomůže, a slovo bude odmítnuto. Druhý pohled je ten, že nedeterministický automat je schopen paralelně provádět mnoho větví výpočtu, a tak najít cestu k přijetí, pokud taková ovšem vůbec existuje. Oba pohledy vystihují naši původní definici nedeterminismu.

Myšlenka nedeterministického přijímání jazyka všech palindromů nad abecedou $A = \{a, b\}$ je poměrně jednoduchá: budeme načítat symboly na zásobník, nedeterministicky uhodneme, jestli právě přišel střed palindromu, načež začneme zásobník vyprazdňovat a kontrolovat, jestli jsou obě poloviny symetrické. Sepišme si to přesně.

Zkonstruujeme NZA přijímající prázdným zásobníkem, jehož množina stavů bude $Q = \{l, p\}$, zásobníkové symboly $Z = \{z_0, a, b\}$, počáteční stav bude l a počáteční zásobníkový symbol z_0. Sada instrukcí bude tato:

$\delta(l, x, z) = (l, zx)\ \forall x \in A, z \in Z$ ... načti levou půlku

$\delta(l, x, z) = (p, zx)\ \forall x \in A, z \in Z$ ... uhodni *sudý* střed

$\delta(l, x, z) = (p, z)\ \forall x \in A, z \in Z$ ... uhodni *lichý* střed

$\delta(p, x, x) = (p, \lambda)\ \forall x \in A$ ... ověř pravou půlku

$\delta(p, \lambda, z_0) = (p, \lambda)$ ... vyprázdni zásobník a skonči

Zjevně pokud automat uhodl střed na správné pozici a obě půlky byly symetrické, slovo bude přijato. Pokud symetrické nebyly, stroj se zastaví před vyprázdněním zásobníku na nedefinovanou instrukci. Pokud stroj uhodl střed palindromu příliš brzo či příliš pozdě, nebude načteno celé slovo nebo se zastaví se na nedefinovanou instrukci před vyprázdněním zásobníku. Tyto větve výpočtu tedy nevydají špatný výsledek a automat tak skutečně přijímá právě jazyk všech palindromů.

Že na tento jazyk nestačí DZA přijímající prázdným zásobníkem se dá odůvodnit téměř stejně, jako v první úloze. Kdyby takový automat přijal palindrom a^i, nemohl by již přijmout palindrom a^{i+1}.

A to je celé.

Tomáš Valla

Úloha 17-4-1 – Mandarinková zeď – program

```
program MandarinkovaZed;
const
  MaxN = 1000;
var
  P: array[1..MaxN] of Integer;
  N: Integer;

  I, Pm, Ps, S: Integer;
begin
  Readln(N);
  for I:= 1 to N do
  Read(P[I]);

  if N = 1 then
    Writeln(P[1])
  else
  begin
    Pm:= P[1] + P[N];
    for I:= 1 to N - 1 do
    if P[I] + P[I + 1] > Pm then
      Pm:= P[I] + P[I + 1];

    if N mod 2 = 0 then
      Writeln(Pm)
```